1 Loops and Conditionals

After learning about the basic objects of Maple, we move on to loops and conditionals. These are necessary for procedural programming where you will write your own functions. Maple provides all the iteration and conditional functions available in any programming language, and a few additional functions that make it more powerful. We start by looking at simple iteration.

The simplest loops are those that do some action when a variable ranges over a sequence, such as an arithmetic progression. This is accomplished using the for loop.

```
> for i from 1 to 4 do
> print(i^2);
> od;
```

$$1$$

$$4$$

$$9$$

$$16$$

The first part of the for statement specifies the range of the variable i. In this case it was from 1 to 4. The default step is 1. The statements to be evaluated for this range are placed between do and od.

do

statements

od;

To increment the variable by some other value, we use the structure,

for i from a to b by c do

(statements)

od;

This basic loop first sets i to a, and evaluates all the statements and then is incremented by c. The statements are repeated, each time increasing i by c, until i is larger than b.

Consider the following example where we build progressions using the loop. We will store the loop values in a list. The list is initialized first, and then we add to it in the loop.

```
> x:=[ ]: for i from 3 to 11 by 3 do
> x:= [ op(x), i]:
> od;
```

$$x := [3]$$

$$x := [3, 6]$$

$$x := [3, 6, 9]$$

```
> x;
```

$$[3, 6, 9]$$

You can suppress the printing of the intermediate results of each iteration by having a colon after od, instead of a semi-colon.

Maple generates the arithmetic progression specified by the loop statements and iterates over this arithmetic progression. This means that if the progression specified is empty, the loop is not evaluated even once. This is important to keep in mind because the behavior is different from that of some other popular languages.

Consider the following example where the specified progression is empty. Our list should come out empty.

```
> x:=[  ]: for i from 3 to 1 by 1  do x:= [op(x), i]; od;
> x;
```

$$[\,]$$

Maple can also iterate over more general sequences than arithmetic progressions. This is accomplished by first constructing a list and using the for statement as follows.

for i in list do

statements

od;

```
> primelist:= [ 2,3, 5, 7,11]:
> for p  in primelist do  print(  p mod 4); od;
```

$$2$$

$$3$$

$$1$$

$$3$$

$$3$$

The for loop syntax is useful to do computations with sequences, lists, and sets.

```
> fiblist:=[1,1,2,3,5,8,13,21,34,55,89];
```

$$\mathit{fiblist} := [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]$$

```
>
> x:=[ ]: for n in fiblist do  x:=[op(x), n mod 3]; od:
> x;
```

$$[1, 1, 2, 0, 2, 2, 1, 0, 1, 1, 2]$$

You can use the same format to iterate over sequences and sets.

```
>
```

The most general iteration is obtained by the use of the while statement. The basic syntax is

while condition do

statements

od;

Suppose we want to find the first prime greater than a specified integer n. We can use the built-in function isprime to check if the number is prime. We can loop through the numbers until we reach a prime using the following list.

```
>
> p:=212:  while not  isprime(p)   do
>   p:=p+1;
> od:
> p;
```

$$223$$

We increment p inside the while loop. The execution of the statement stops as soon as the condition becomes false. In this case it is not isprime, hence the loop will stop when p is prime.

Exercise: Find the first prime above 8765930291. What happens in the loop above if the starting value of p is prime.

We will see how to combine conditions in the following so that more complicated conditions can be used to control the flow of the program using a while loop.

Logical Statements.

The flow control of a program is accomplished by the if statement. The format is

if condition

then statement

else statement.

The if statements end with fi, just as the do statement ends with an od.

We use this inside a loop to print the primes using the isprime function.

```
> for p from 100 to 110  do
> if isprime(p) then print(p) fi;
> od;
```

$$101$$

$$103$$

$$107$$

$$109$$

```
>
```

In the following example, we use the if statement and the branching possibilities to construct two lists, one containing primes p such that p mod 4 is 1 and the other those for which p mod 4 is 3.

```
> list1:= [ ]: list2 := [ ]:
> for i from 2 to 100 do
> if   modp(i, 4) =1 then list1:= [ op(list1), i];
> else if modp(i,4) =3 then list2:= [op(list2), i] ;
> fi;
> fi;
> od;
> list1;
```

$$[5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73, 77, 81, 85, 89,\\ 93, 97]$$

```
> list2;
```

$$[3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63, 67, 71, 75, 79, 83, 87,\\ 91, 95, 99]$$

As another example, consider the following loop where we print primes of the form $x^2 + y^2$. We can use two loops and an if statement as follows.

```
> for x from 1 to 10 do
> for y from x to 10 do
> if isprime(x^2+y^2) then print(x^2+y^2);
```

```
> fi; od;od;
```

2

5

17

37

101

13

29

53

73

109

41

97

61

89

113

149

181

This list generated the primes of the form $x^2 + y^2$. It is equally to simple to combine loops and if statements to do more general operations.

Exercise: Modify the loop above to print the remainder when a prime of the form $x^2 + y^2$ is divided by 4.

Conditions used in the if statement can be combined by the logical operations and, or, not. The logical true and false values are represented in Maple by true and false.

Let us test the conditionals and their logical combinations.

```
> x:=11:
```

This sets x to 11. To check the value, we can use = for equality testing. The function evalb is used to make Maple evaluate statements immediately.

```
> evalb( x=11);
```

true

```
> evalb(x=10);
```

false

```
> isprime(x);
```

true

Exercise: Write a loop to print the primes between 2 and 100 of the form 8n+1. Do this a) using a for loop and b) a while loop?